Les types primitifs et les classes enveloppes

par Anis Frikha (Mon site)

Date de publication : 15/06/2006

Dernière mise à jour :

Ce tutoriel vise à présenter les différents types primitifs de Java ainsi que les classes enveloppes. Il indique comment bien les employer en montrant les mécanismes qui se cachent derrière certaines opérations.

I - Introduction
II - Les types primitifs
II-A - Les types primitifs entiers
II-B - Les types primitifs flottants
II-C - Le type primitif caractère
II-D - Le type primitif booléen
II-E - Le type primitif void
II-F - Emploi des types primitifs
III - Les classes enveloppes
III-A - Les classes enveloppes numériques
III-B - La classe enveloppe Character
III-C - La classe enveloppe Boolean
IV - Récapitulatif
V - Nouveautés Java 5
VI - Conclusion
VI - Remerciement
VIII - Liens utiles

Copyright © 2007 - Anis Frikha. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

I - Introduction

Ce tutoriel vise à présenter les différents types primitifs de java ainsi que les classes enveloppes. Il indique comment bien les employer en montrant les **mécanismes** qui se cachent derrière certaines opérations. Certains se poseront la question : **Pourquoi un article sur un sujet aussi banal ?** La réponse est que les types primitifs sont une exception en Java car ce ne sont pas des objets, ce qui rend Java (si on voulait être rigoureux) un language non pas orienté objet au sens de SmallTalk, mais un language applicant une **"philosophie" objet**. De plus, connaître les intervalles de validité des types primitifs est capital: pour rappel un crash de la fusée Ariane a été causé par un "simple" dépassement de capacité ! Maintenant que vous avez compris (je l'éspère :)) l'importance du sujet, allons à la découverte de ces fameux types primitifs.

Copyright © 2007 - Anis Frikha. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

II - Les types primitifs

Les types primitifs en Java sont: *byte, short, int, long* pour les types entiers, *float, double* pour les "réels", un seul type caractère qui est *char*, et un type booléen à savoir *boolean*. Notez le fait que tous ces types ont leur première lettre miniscule, ça montre que ce ne sont pas des classes d'après les conventions de nommage en Java.

II-A - Les types primitifs entiers

Les types entiers servent à représenter des entiers relatifs: il s'agit des types byte, short, int, et long, dont voici un tableau récapitulant leurs tailles, leurs maximums et minimums:

Type	Taille (en octet)	Valeur minimale	Valeur maximale
byte	1	-128	127
short	2	-32 768	32 767
int	4	-2 147 483 648	2 147 483 647
long	8	-9 223 372 036 854 775 808	9 223 372 036 854 775 807

Intéressons-nous à présent à la représentation en mémoire de ces types. Pour cela, on doit faire la distinction entre les nombres positifs et négatifs. En effet, un nombre positif est codé de la manière suivante:

- Le bit de poids fort est appelé le bit de signe et il a la valeur 0.
- Les autres bits représentent la valeur absolue du nombre codé en base 2.

Pour les nombres négatifs en revanche, la représentation est comme suit:

- Le bit de poids fort qui est le bit de signe vaut 1.
- La valeur absolue du nombre est codée par la technique du complément à deux.

Mais qu'est-ce que la technique du complément à deux ?

La technique du complément à deux consiste à :

- Inverser tous les bits dans la representaion binaire de la valeur absolue du nombre : c'est à dire remplacer tous les 0 par des 1 et les 1 par des 0.
- Ajouter 1 au nombre obtenu.

Avec cette technique, additionner/soustraire deux nombres entiers relatifs se résume à additionner bit à bit. Cette opération nécessite donc très peu de circuits logiques au sein de l'UAL (Unité Arithmétique et Logique).Il en résulte des calculs très rapides.

Voici un petit exemple pour assimiler la chose.On s'intéresse à l'opération : 45-63, ce qui est equivalent à 45+(-63) et on supposera que ces nombres sont des bytes donc codé sur un seul octet. La représentation binaire de **45** est :**00101101**. Pour la representation binaire de **-63**, on adopte la démarche suivante:

Representation de la valeur absolue	**00111111**

Copyright © 2007 - Anis Frikha. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

Inversion des bits	**11000000**
Complément à deux	**11000001**

Il nous reste plus maintenant qu'à faire l'addition bit à bit:

45	**00101101**
-63	**11000001**
-18	**11101110**

Comme le résultat de cette addition est un nombre négatif, il sera sous la forme "complément à deux", , ce qui est fort pratique pour le processeur afin de réaliser des opérations successives.

Et voilà, c'est magique , non ?

Intéressons nous maintenant aux régles de conversion. Il est bien entendu possible de convertir une valeur codée dans un type donné vers un type de plus grande capacité et ceci sans perte de précision. On peut imaginer la conversion comme ceci:

byte-->short-->int-->long

On peut affecter une valeur entière à une variable de différentes façons : supposons que a soit de type int et qu'on veuille lui affecter la valeur 33, on peut procéder comme suit:

- a=33; c'est la manière classique de faire, on utilise une notation **décimale**
- a=0x21; on utilise une notation **héxadécimale**, pour cela on fait préceder la valeur en héxadécimal par **0x**
- a=041; on utilise ici une notation **octale**, pour cela on fait précédé la valeur en octal par **0**
- a=33L ici on force 21 à étre de type long en la faisant suivre d'un l ou L.Notez que pour des raisons de visibilité il vaut mieux utiliser une majuscule.

II-B - Les types primitifs flottants

Il existe en Java deux types primitifs permettant de représenter des nombres flottants, à savoir *float* et *double*, dont voici les valeurs limites:

Type	Taille (en octets)	Valeur minimale	Valeur maximale
float	4	-1.40239846E-45	3.40282347E38
double	8	4.94065645841246544E-324	1.797693134862316E308

Si on exécute l'opération 0.3-(3*0.1) le résultat ne sera pas nul !! Une petite explication s'impose. En fait la raison réside dans la représentation en mémoire des nombres flottants de type *double* : elle est composée d'un bit de signe, de la mantisse sur 52 bits et de l'exposant sur 11 bits. Pour le type *float*, il est représenté par un bit de signe, une mantisse sur 23 bits et un exposant sur 8 bits.

Le nombre est alors représenté sous la forme d'un rationnel approchant au maximum le réel entré.

Copyright © 2007 - Anis Frikha. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

*Seuls les entiers et les rationnels dont le dénominateur est une puissance de 2 peuvent être représentés exactement.Si vous avez besoin d'une grande précision lors de vos calculs, je vous conseille de recourir à la classe **BigDecimal***

*Que ce soit pour les entiers ou pour les flottants, il n'est pas nécessaire de connaître par coeur leurs valeurs limites, par contre il est important d'avoir une idée sur leur ordre de grandeur car **les dépassements de capacité ne sont jamais signalés et ce quelque soit le language**.*

Il existe différentes façons d'affecter une valeur à une variable flottante. Supposons que a et b soient respectivement de type float et double. Les affectations suivantes sont correctes :

- a=15f le **f** est obligatoire pour signaler que c'est un *float*
- b= 0.12 c'est la notation **décimale**
- b=4.25E4 qui est equivalent à b=4.25e4 c'est la notation **exponentielle** ou **scientifique**.

NOTE : On peut forcer une valeur à être de type double en faisant suivre sa valeur par la lettre D ou d .

II-C - Le type primitif caractère

Le type caractère en Java est représenté par *char* qui est codé, non pas sur un octet comme la plupart des autres languages, mais sur **deux octets** afin de supporter l'**Unicode**. Outre les caractères qu'on peut saisir au clavier, on peut entrer d'autres caractères non disponible sur le clavier en passant par le code Unicode en hexadécimal du caractère et ceci avec la syntaxe suivante **\uxxxx** où xxxx représente le code. Par exemple le caractère pi a pour code 03C0.

Le type char est un peu particulier dans la mesure où on peut affecter **une valeur entière positive** à une variable de type char.

*Le type char est le **seul** à n'accepter que des valeures entières **positives***

L'exemple de code suivant permet d'afficher le symbole PI :

```
public class TestChar{
 public static void main(String args[]){
  char c=1055;
  System.out.println(c);
 }
}
```

Le type *char* pouvant accepter des valeures numériques, il possède lui aussi une capacité et des limites représentées par le tableau ci-dessous.

Taille (en octets)	Valeur minimale	Valeur maximale
2	0	65 536

II-D - Le type primitif booléen

Contrairement à d'autre languages, Java dispose d'un type booléen, il s'agit de *boolean*.On peut par exemple affecter le resultat d'un test à une variable de type *boolean* ainsi l'expression:

Copyright © 2007 - Anis Frikha. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

```
boolean test=(5==3);
```

a un sens.

Une variable de type *boolean* ne peut prendre que deux valeurs à savoir : *true* et *false*.

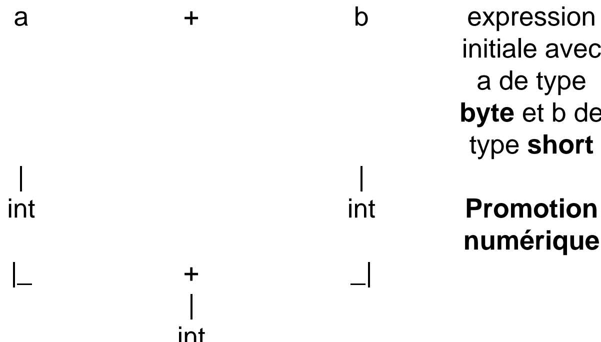
II-E - Le type primitif void

Ce type n'a pas d'intêret majeur mis à part le fait qu'il permet de signaler qu'une méthode ne renvoie rien.

II-F - Emploi des types primitifs

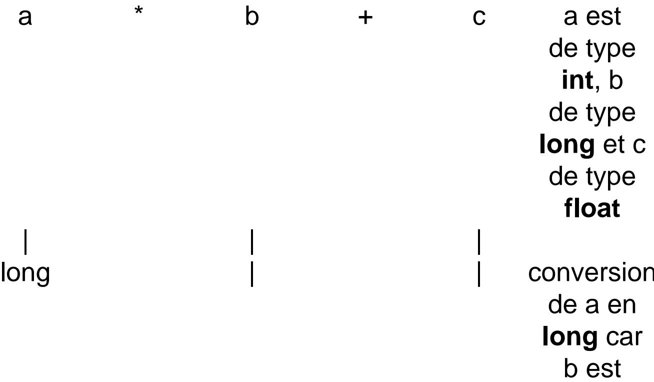
Java comme la plupart des autres languages de programmation dispose d'opérateurs arithmétiques binaires c'est à dire portant sur deux opérandes. Ces opérateurs sont : + (addition), - (soustraction), * (multiplication), / (division) , et enfin % (modulo). Ces opérateurs ne sont pas définis pour les types *byte*, *short* et *char*, pourtant il est possible d'utiliser des variables de ces types dans des expressions arithmétiques grâce à **la promotion numérique**.

La promotion numérique consiste à changer ces types non définis en int. Par exemple :

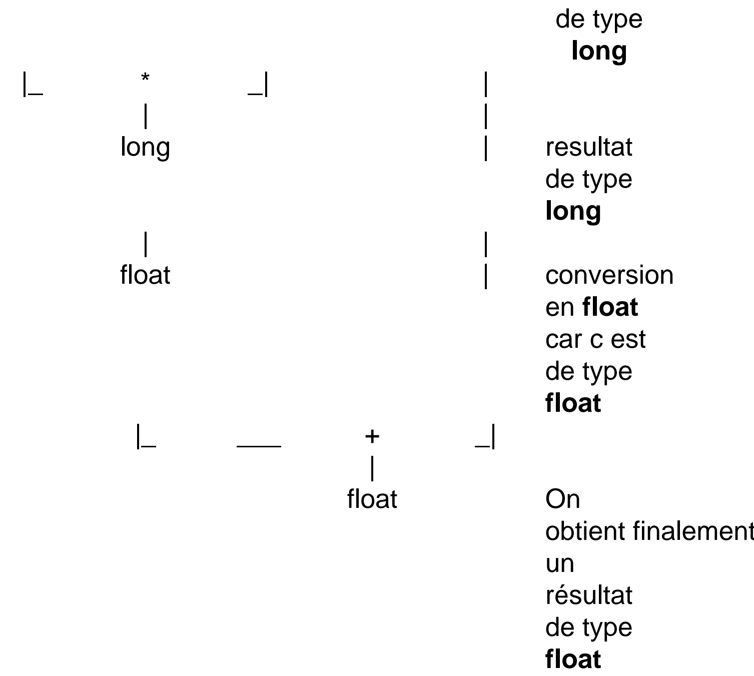


On peut aussi s'interroger sur le fonctionnement des ces opérateurs lorsque les opérandes ne sont pas de même type.En fait, dans ce cas , on a affaire à ce qu'on appelle **les conversions d'ajustement de type**.Java procède à un changement du type des opérandes afin qu'elles soient toutes de même type pour pouvoir calculer l'expression. Ce changement de type se fait de telle sorte qu'il n'y ait pas de perte de précision, c'est à dire suivant la hiéarchie suivante : int->long->float->double.

Considérons l'exemple suivant :



Copyright © 2007 - Anis Frikha. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.



En Java, on peut faire intervenir des variables de type *char* dans une expression arithmétique grâce à la promotion numérique en *int*. Ceci est possible car une variable de type *char* n'est qu'une suite de bits en mémoire qui peut étre vue comme:

- Un entier de type int
- Le code Unicode d'un caractère

ainsi une expression du type a+b où a et b sont de type *char* aura pour résultat un *int*. Et la conversion char vers int est tout à fait licite : Si i est de type *int* et j de type *char*, faire i=j est correct.

Si on désire "casser" la chaîne de conversion , c'est à dire changer de sens , la seule manière d'y arriver est de faire un cast. Attention à la perte de précision ! Par exemple avec a un short et b un int, on peut faire a=(short)b. Si b est trop grand pour tenir sur 2 octets (qui est la taille de short) alors les bits en trop seront perdus.

⚠ *Bien que le type short a une taille de 2 octets, on ne peut pas affecter, sans cast, un char à un short. Il en va de mème pour le type byte.A noter que les conversions sans perte de précision (et donc sans cast) sont tout à fait possibles du moment que l'on respecte l'ordre de grandeur des types : byte < char|short < int < long < float < double*

Les classes Double et Float possédent quelques attributs statiques intéressants qui sont : POSITIVE_INFINITY, NEGATIVE_INFINITY et NaN (Not a Number). Voici quelques cas d'utilisation :

- 1+Double.POSITIVE_INFINITY renvoie POSITIVE_INFINITY
- Double.POSITIVE_INFINITY+Double.POSITIVE_INFINITY renvoie aussi POSITIVE_INFINITY
- 12/0 renvoie Double.POSITIVE_INFINITY
- 0/0 renvoie Double.NaN
- Double.POSITIVE_INFINITY+Double.NEGATIVE_INFINITY renvoie Double.NaN

Copyright © 2007 - Anis Frikha. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

III - Les classes enveloppes

Les classes enveloppes (appelées aussi *wrappers*) sont nées d'un besoin de pouvoir encapsuler des types primitifs dans des objets afin de pouvoir par exemple les mettre dans une collection ou bien dans une base de donnée objet,

III-A - Les classes enveloppes numériques

Les classes enveloppes numériques dérivent toutes de la classe *Number* et elles sont : *Byte*, *Short*, *Integer* , *Long*, *Float* et enfin *Double*.

Toutes ces classes admettent un constructeur ayant pour paramètre le type primitif correspondant. Par exemple pour obtenir un objet *Integer* à partir d'un int de valeur 2 il suffit de faire : *Integer objetEntier=new Integer(2);*

Ces classes présentent diverses méthodes semblables dont voici une présentation des plus utilisées à mon avis :

- Les méthodes **byteValue()**, **shortValue()**, **intValue()**, **longValue()**, **floatValue()** et **doubleValue()** : Ces méthodes,présentes dans toutes les classes numériques (car héritées de la super classe Number), renvoient le type primitif indiqué. Elles constituent donc un pont entre les objets et les types primitifs.
- Les méthodes statiques **parseByte(String)**, **parseShort(String)**, **parseInt(String)**, ...:qui renvoient le type **primitif** correspondant. Attention ces méthodes peuvent générer une exception du type *NumberFormatException* et doivent donc étre placées dans un bloc try/catch.
- Les méthodes statiques **valueOf(String)** qui renvoient elles en revanche **un objet**. Par exemple *Integer.valueOf("4");* renverra un objet de type *Integer* ayant pour valeur 4.

Voici des exemples de code illustrant l'emploi ces méthodes:

La méthode parseInt(String):

```
public class Parse{
 public static void main(String args[]){
  String nbreChaine="455";
  int vraiNbre;
  try{
   vraiNbre=Integer.parseInt(nbreChaine);
  }catch(NumberFormatException e){
   System.out.println("nbreChaine n'est pas un vrai nombre");
  }
 }
}
```

La méthode parseInt(String,int) où int représente la base de la chaîne en argument :

- parseInt("473", 10) retourne 473 car on considère la chaîne comme étant codée en **base 10**.
- parseInt("-FF", 16) retourne -255 ici on emploie la **base 16**.
- parseInt("1100110", 2) retourne 102 car la chaîne est codée en **base 2**.

Les méthodes valueOf(String, int) ont le même comportement que les parseXxx(String,int) sauf qu'elles **retournent des objets** et non des types primitifs.

Copyright © 2007 - Anis Frikha. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

Si aucune base n'est spécifiée, c'est à dire qu'on utilise les méthodes parse ou valueOf à un seul paramètre, la base par défaut sera la base 10.

Ainsi *parseInt("473", 10)* sera équivalent à *parseInt("473")*

À noter que toutes ces classes,en plus de la classe Character, possédent des attributs MIN_VALUE et MAX_VALUE representant les valeurs limites pour chaque type,ainsi qu'un attribut SIZE indiquant le nombre de bits nécessaires afin de représenter le nombre.

III-B - La classe enveloppe Character

Cette classe sert comme vous vous en doutez à encapsuler un char. Elle dispose d'un grand nombre de méthodes permettant de savoir si c'est une majuscule ou une miniscule, de retrouver le code Unicode du caractère,...Une consultation de la **javadoc** vous donnera une vue exhaustive des capacités de cette classe.

Character ***n'héritant pas de Number*** *ne dispose pas de méthodes telles que charValue ou parseChar*

III-C - La classe enveloppe Boolean

Bien que cette classe ne présente pas un intêret majeur, il est tout de mème utile de connaitre son existance et d'avoir regarder la page **javadoc** la concernant au moins une fois ;)

Copyright © 2007 - Anis Frikha. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

IV - Récapitulatif

Type primitif	Objet associé
int	Integer
char	Character
float	Float
double	Double
long	Long
boolean	Boolean
byte	Byte
short	Short
void	Void

toutes ces classes redéfinissent la méthode toString() héritée de Object afin qu'elle renvoie le type primitif en question sous forme de chaine de caractère.

les classes enveloppes sont immuables, et toutes modifications de la valeur aboutissent à un nouvel objet.

Copyright © 2007 - Anis Frikha. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

V - Nouveautés Java 5

Avec la version 5 de Java, est apparue une nouveauté fort pratique, à savoir le **boxing** et l'**unboxing**. Cela consiste à pouvoir passer aisément des types primitifs vers les classes enveloppes et des classes enveloppes vers les types primitifs. Par exemple, ce bout de code est tout à fait correct :

```
int a=6;
Integer b=new Integer(0);
b=a;
```

Ici b aura la valeur de a c'est à dire 6. C'est ce qu'on appelle le **boxing**

L'inverse est aussi possible :

```
a=b;
```

à ce moment là a vaudra 0.IL s'agit de l'**unboxing**

Ce mécanisme est très pratique notamment lors de l'usage des collections contenant des types primitifs. C'est un simple appel implicite de méthodes :

```
Integer integer = 0;
```

est équivalent à :

```
Integer integer = Integer.valueOf(0);
```

, et

```
int a = integer;
```

est lui équivalent à :

```
int a = integer.intValue();
```

Autre nouveauté de Java 5 : la méthode *decode* qui est présente dans toutes les classes enveloppes numériques. C'est une méthode statique qui retourne un objet du type correspondant depuis une chaine représentant un nombre écrit en base 8,10,ou 16. Ainsi:

- decode("012") retourne 10, car la chaine commence par 0, elle donc bien écrite en **base 8**
- decode("0x12") retourne 18, car la chaine est lue en **base 16** à cause du 0x
- decode("12") retourne bien sùr 12 car la chaine est déjà en **base 10**

Copyright © 2007 - Anis Frikha. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

On peut ausi citer, comme autre nouveauté, la méthode statique reverseBytes(xxx) présente dans toutes les classes enveloppes mis à part void. Cette méthode retourne le complément à deux du type primitif xxx, le résultat est bien entendu du mème type que l'argument.

À noter aussi parmi les nouveautés la possibilité d'avoir un équivalent héxadécimal pour la notation scientifique. Ainsi double d = 0x10P4 ferra que d = 256 .

Copyright © 2007 - Anis Frikha. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

VI - Conclusion

Voilà ce petit tutorial touche à sa fin. J'espère qu'il vous a été utile et qu'il vous a permis d'assimiler cet aspect de Java. Sur ce, je vous souhaite bon code :) .

Copyright © 2007 - Anis Frikha. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

VI - Remerciement

Je remercie toute l'équipe Java et particulièrement **vbrabant**, **fearYourSelf**, **adiGuba** et **hiko-seijuro** pour leur relecture et corrections.

Copyright © 2007 - Anis Frikha. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

VIII - Liens utiles

- L'incontournable **javadoc**
- Un **article** fort intéressant sur la réprésentation du type double en mémoire
- Un **article** fait maison sur l'autoboxing.

Copyright © 2007 - Anis Frikha. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

Copyright © 2007 - Anis Frikha. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.